We claim:

1. In a multitasking operating system that uses virtual memory to share physical memory among concurrently executing application programs, a method for controlling allocation of physical memory comprising:

in response to a call from an application program to group specified code or data in a group, creating a structure to group the code or data specified by the application;

monitoring for a not-present interrupt generated in response to request to access any part of the code or data in the group;

when the not-present interrupt occurs for a unit of memory in the group, loading all of the code or data in the group that is not already in physical memory into physical memory from secondary storage at one time, including loading the unit of memory for which the not present interrupt has occurred and all other units of memory used to store the code or data in the group.

2. The method of claim 1 wherein the structure includes a linked list structure that links together code or data stored at non-contiguous portions of virtual memory.

3. The method of claim 2 wherein the structure links pages of memory associated with the non-contiguous portions of code or data.

4. The method of claim 1 further including:

repeating the steps of claim 1 for additional groups of code or data specified by the application.

5. The method of claim 4 further including:

repeating the steps of claim 1 for a group of code or data for another concurrently executing application such that more than one concurrently executing application program has specified at least one group of code or data to be treated as a single piece of memory for loading into physical memory in response to a not-present interrupt.

6. The method of claim 1 further including:

when the not-present interrupt occurs, checking whether the interrupt has occurred for a unit of memory in the group by evaluating whether an address of the memory request for which the interrupt occurred is within a series of non-contiguous memory addresses of the group.

7. The method of claim 1 further including:

tracking memory accesses to units of memory in the group together such that when a unit of memory in the group is accessed, all of the units of memory in the group are marked as accessed; and

determining which portions of physical memory to swap from physical memory to secondary storage by determining which units of code are marked as accessed, such that units are selected to be swapped from physical memory to secondary storage based on frequency of use or how recently the units of code have been accessed.

8. The method of claim 7 further including:

in response to a call from an application program to group specified code or data in a second group, creating a second structure to group the code or data specified by the application;

tracking memory accesses to units of memory in the first and second group such that when a unit of memory in both the first and second group is accessed, all of the units of memory in the first and second group are marked as accessed and the unit of memory in both the first and second group is marked as being accessed twice.

9. The method of claim 8

when a block of code or data shared between two or more groups is accessed, marking the block as being accessed n times where n is the number of groups that share the block.

10. A computer-readable medium storing instructions for performing the steps of claim 1.

11. In a multitasking operating system that uses virtual memory to share physical memory among concurrently executing application programs, a virtual memory management system comprising:

a physical memory manager for swapping code and data between secondary storage and physical memory to enable applications to share physical memory and for loading units of memory from secondary storage to physical memory;

an API module for grouping portions of code or data in a group in response to a function call from an application program that designates portions of the code or data to be put in the group; and

a memory monitor in communication with the physical memory manager and the API module, the memory monitor operable to monitor a processor for a not-present interrupt, and operable to invoke the physical memory manager to load in all portions of the group not already in physical memory when the processor generates a not-present interrupt in response to a memory request directed to any code or data in the group.

12. The virtual memory management system of claim 11 wherein the memory monitor is operable to track memory accesses to units of memory in a group and is operable to mark all units of memory as accessed when any one of the units of memory in a group is accessed.

13. The virtual memory management system of claim 11 wherein the memory monitor is operable to track memory accesses to blocks of memory that are shared among more than one group and is operable to mark all units of memory in a shared block n times when any one of the units of memory in the shared block is accessed, where n is the number of groups that include the shared block.

14. The virtual memory manager of claim 11 wherein the physical memory manager is operable to monitor memory accesses to units of memory and is operable to swap units of memory from physical memory to secondary storage when necessary to satisfy a request for a piece of code or data that is not present in physical memory.

15. The virtual memory manager of claim 14 wherein the units of memory are pages.

16. The virtual memory manager of claim 14 wherein the physical memory manager marks a unit of memory as used in response to a memory request for the unit; wherein the physical memory manager marks all units of memory in a group as used whenever a unit of memory in the group is accessed, and wherein the physical memory manager selects which of the units to swap from physical memory to secondary storage to satisfy a memory request for physical memory by determining which unit or units are least recently used.

17. The virtual memory manager of claim 11 wherein the API module is responsive to concurrently executing application programs and is operable to maintain data structures representing groups of code or data for more than one application program to be loaded into physical memory together in response to a not-present interrupt for a unit of memory that resides in one or more of the groups.

18. The virtual memory manager of claim 11 wherein the API module is operable to enable the application program to add or delete code or data from the group dynamically, at run time.

19. The virtual memory manager of claim 18 wherein the API module is operable to create and dynamically update a data structure maintaining a list of memory blocks included in the group based on requests by the application to create the group and change the code or data in the group.

20. A computer-readable medium having stored thereon a data structure comprising:

a series of data fields representing blocks of code or data associated with an application to be treated as a single unit for purposes of virtual memory management, the data fields including a list of memory addresses of the blocks and sizes of each block in the list;

wherein the data structure is evaluated in a data processing operation to load each of the blocks into physical memory whenever a not-present interrupt is generated for any memory address referring to a location included in one of the blocks.